In the Claims:

1. (Currently amended) A computer readable storage medium storing program instructions executable by a processor of a multithreaded system to implement a mutual exclusion mechanism comprising:

a biasable lock that is concurrently accessible to a plurality of threads on the multithreaded system and is biased to at most one of the plurality of threads at a given time, wherein the biasable lock includes a quick-lock field;

acquisition and release sequences that, when executed by a bias-holding thread on the processor, wherein the biasable lock is biased to the bias-holding thread, are free of atomic read-modify-write operations and wherein the acquisition sequence comprises:

storing a value in the quick-lock field, the value indicating that the biasable lock is held by the bias-holding thread; and

subsequent to said storing, determining whether the biasable lock is still biased to the bias-holding thread, wherein the acquisition sequence completes successfully and free of atomic read-modify-write operations if the biasable lock is determined to still be biased to the bias-holding thread

~~and of memory barrier operations executable to constrain the processor from performing at least one type of instruction reordering; and~~

~~wherein the processor supports out-of-order instruction execution~~.

2. (Canceled)

3. (Previously presented) The computer readable storage medium of claim 1, wherein the acquisition and release sequences include only read and write operations when executed by the bias-holding thread.

4. (Previously presented) The computer readable storage medium of claim 1, wherein the biasable lock is initially unbiased.

5. (Previously presented) The computer readable storage medium of claim 1, wherein the biasable lock is biased on creation.

6. (Previously presented) The computer readable storage medium of claim 1, wherein the biasable lock is biased on acquisition.

7. (Currently amended) The computer readable storage medium of claim 1, wherein the bias-holding thread is not the ~~creating~~ thread that created the lock.

8. (Previously presented) The computer readable storage medium of claim 1, wherein execution of the acquisition sequence by the bias-holding thread employs a programming construct that precludes reordering of a particular read before a particular write.

9. (Currently amended) The computer readable storage medium of claim 8, wherein the precluded reordering includes reordering by a compiler.

10. (Previously presented) The computer readable storage medium of claim 8, wherein the precluded reordering includes reordering by the processor.

11. (Previously presented) The computer readable storage medium of claim 8, wherein the programming construct employs collocation of the target of the particular read and the target of the particular write.

12. – 16. (Canceled)

17. (Currently amended) The computer readable storage medium of claim 8,

wherein the particular read loads a lock status as part of said determining whether the biasable lock is still biased to the bias-holding thread; and

wherein the particular write stores [[a]] the value in the quick-lock field ~~indication~~.

18. (Previously presented) The computer readable storage medium of claim 8, wherein the preclusion is based, at least in part, on characteristics of an implementation of a memory model implemented by the processor.

19. (Previously presented) The computer readable storage medium of claim 1,

the processor implements a total store order (TSO) memory model,

wherein the acquisition by the bias-holding lock includes a store operation followed in program order by a load operation, the store and load operations having collocated targets that encode a quick lock indication and lock status, respectively.

20. (Previously presented) The computer readable storage medium of claim 1, wherein the biasable lock includes an MCS lock augmented to provide fast path acquisition and release sequences for the bias-holding thread.

21. (Currently amended) The computer readable storage medium of claim 1, wherein the biasable lock includes an TAT[[ ]]AS lock augmented to provide fast path acquisition and release sequences for the bias-holding thread.

22. (Previously presented) The computer readable storage medium of claim 1, wherein the biasable lock includes a lock provided by a POSIX pthreads mutex library, augmented to provide fast path acquisition and release sequences for the bias-holding thread.

23. (Previously presented) The computer readable storage medium of claim 1, wherein the biasable lock includes a monitor provided by a Java virtual machine implementation, the monitor augmented to provide fast path acquisition and release sequences for the bias-holding thread.

24. (Previously presented) The computer readable storage medium of claim 1, wherein the biasable lock is rebiasable to another thread during course of a computation that employs the lock.

25. (Previously presented) The computer readable storage medium of claim 1, wherein bias of the lock to the bias-holding thread is revocable and revocation of bias by, or on behalf of, a contending thread, is mediated, at least in part, using a signal handler.

26. (Previously presented) The computer readable storage medium of claim 1, wherein bias of the lock to the bias-holding thread is revocable and revocation of bias by, or on behalf of, a contending thread, is mediated, at least in part, using a cross-call.

27. (Previously presented) The computer readable storage medium of claim 1, wherein bias of the lock to the bias-holding thread is revocable and revocation of bias by, or on behalf of, a contending thread, is handled, at least in part, at a garbage collection safe point.

28. (Currently amended) The computer readable storage medium of claim 24, wherein the program instructions are further executable to implement:

detecting a current level of contention;

~~revoking the bias of the biasable lock in response to detecting a given level of contention;~~ and

a rebiasing sequence executable by the processor to rebias the lock to the thread in response to detecting a lower level of contention.

29. (Previously presented) The computer readable storage medium of claim 28, wherein the detecting a current level of contention detection comprises accessing a request queue to identify the level of contention.

30. (Previously presented) The computer readable storage medium of claim 28, wherein the detecting a current level of contention detection comprises employing an attempt counter to identify the level of contention.

31. (Currently amended) A computer readable storage medium storing program instructions executable by a processor of a multithreaded system to implement a biasable lock wherein:

the bias able lock is concurrently accessible to a plurality of threads on the multithreaded system and is biased to at most one of the plurality of threads at a given time, wherein the biasable lock includes a quick-lock field;

the biasable lock provides at least two acquisition sequences including a fast path acquisition sequence for a bias-holding thread to which the biasable lock has been biased and a second acquisition sequence, the fast path acquisition sequence optimized relative to the second acquisition sequence;

wherein the fast path acquisition sequence is free of atomic read-modify-write operations and comprises:

storing a value in the quick-lock field, the value indicating that the biasable lock is held by the bias-holding thread; and

subsequent to said storing, determining whether the biasable lock is still biased to the bias-holding thread, wherein the acquisition sequence completes successfully and free of atomic read-modify-write operations if the biasable lock is determined to still be biased to the bias-holding thread

~~and of explicit memory barrier operations executable to constrain the processor from performing at least one type of instruction reordering; and~~

~~wherein the processor supports out-of-order instruction execution~~.

32. (Canceled)

33. (Canceled)

34. (Previously presented) The computer readable storage medium of claim 31, wherein the second acquisition sequence implements one of an MCS lock, a TAT AS lock, a lock consistent with that provided by a POSIX pthread Mutex library and a Java monitor.

35. (Previously presented) The computer readable storage medium of claim 31, wherein the biasable lock is rebiasable.

36. (Currently amended) A computer-implemented method, the method comprising:

instantiating a biasable lock that is concurrently accessible to a plurality of threads on the multithreaded system and is biased to at most one of the plurality of threads at a given time, wherein the biasable lock includes a quick-lock field; and

for the thread to which bias has been directed, releasing and acquiring the biasable lock using fast path instruction sequences that are free of atomic read-modify-write operations wherein the acquisition sequence comprises:

storing a value in the quick-lock field, the value indicating that the biasable lock is held by the bias-holding thread; and

subsequent to said storing, determining whether the biasable lock is still biased to the bias-holding thread, wherein the acquisition sequence completes successfully and free of atomic read-modify-write operations if the biasable lock is determined to still be biased to the bias-holding thread

~~and of explicit memory barrier operations executable to constrain the processor from performing at least one type of instruction reordering~~.

37. (Original) The method of claim 36, further comprising: directing the bias to the thread coincident with a first acquisition of the biasable lock.

38. (Original) The method of claim 36, further comprising: directing the bias to the thread coincident with the instantiation.

39. (Original) The method of claim 36, further comprising: directing the bias to the thread coincident with creation of an object.

40. (Original) The method of claim 36, wherein directing of bias to the thread is performed by another thread.

41. (Previously presented) The method of claim 36, further comprising: for a thread other than the thread to which bias has been directed, acquiring the lock using an instruction sequence that unbiases the lock, if it is biased.

42. (Previously presented) The method of claim 36, further comprising: rebiasing the biasable lock to another thread.

43. (Previously presented) The method of claim 36, further comprising: executing the program code as a multi-threaded software application configured to utilize the biasable lock to allow the bias-holding thread of the multi-threaded software application to repeatedly acquire and release the biasable lock, at least in part by executing one or more of the fast path instruction sequences.

44. (Currently amended) The method of claim 43, further comprising: after rebiasing to another thread, allowing the another thread to repeatedly acquire and release the lock without executing any atomic operations ~~with extremely low overhead~~.

45. (Previously presented) The method of claim 36, wherein the method is performed as part of executing program code in a single-threaded execution environment,

wherein the program code is compiled with the biasable lock for execution on both the single-threaded execution environment and on a multi-threaded execution environment, and

wherein the biasable lock allows the program code to run in the single-threaded execution environment without significant lock-related overhead.

46. (Currently amended) A computer readable storage medium storing program instructions executable by a processor of a multithreaded system to implement:

a biasable software lock that ~~that~~ is concurrently accessible to a plurality of threads on the multithreaded system and is biased to at most one of the plurality of threads at a given time and maintains both a lock state indicated by a value of a quick-lock field and a bias state, whereby acquisition of the lock by a thread to which bias has been directed is more efficient than acquisition of the lock by another thread and does not include any atomic read-modify-write operations and comprises:

storing a value in the quick-lock field, the value indicating that the biasable software lock is held by the thread to which bias has been directed; and

subsequent to said storing, determining whether bias is still directed to the thread, wherein the acquisition completes successfully and free of atomic read-modify-write operations if the bias is still directed to the thread

~~or memory barrier operations executable to constrain the processor from performing at least one type of instruction reordering; and~~

~~wherein the processor supports out-of-order execution~~.

47. (Previously presented) The computer readable storage medium of claim 46, wherein the lock is rebiasable to another thread during course of a computation that employs the lock.

48. (Previously presented) The computer readable storage medium of claim 46, including acquisition and release sequences that, when executed by the thread to which bias has been directed, include only read and write operations.

49. (Canceled)

50. (Canceled)

51. (Previously presented) The computer readable storage medium of claim 48, wherein the acquisition sequence employs a programming construct that precludes reordering of a particular read before a particular write.

52. (Previously presented) A computer readable storage medium storing program instructions executable by one or more processors of a multithreaded system to implement a program product including a mutual exclusion mechanism embodied therein, the program product comprising:

a data structure instantiable in memory of a processor to implement a lock that includes a bias attribute, wherein the lock is concurrently accessible to a plurality of threads on the multithreaded system and is biased to at most one of the plurality of threads at a given time, wherein the biasable lock includes a quick-lock field;

a lock acquisition sequence of operations executable by the processor, the lock acquisition sequence having a fast path for a thread to which bias has been directed and a second path, the fast path optimized relative to the second path;

wherein the fast path acquisition sequence is free of atomic read-modify-write operations and comprises:

storing a value in the quick-lock field, the value indicating that the lock is held by the thread to which bias has been directed; and

subsequent to said storing, determining whether bias is still directed to the thread, wherein the acquisition completes successfully and free of atomic read-modify-write operations if the bias is still directed to the thread

~~and of explicit memory barrier operations executable to constrain the processor from performing at least one type of instruction reordering; and~~

~~wherein the processor supports out-of-order instruction execution~~.

53. (Canceled)

54. (Canceled)

55. (Previously presented) The computer readable storage medium of claim 52, further comprising: a lock release sequence of operations executable by the processor, the lock release sequence having a fast path for the thread to which bias has been directed and a second path, the fast path optimized relative to the second path.

56. (Previously presented) The computer readable storage medium of claim 52, wherein the acquisition sequence employs a programming construct that precludes reordering of a particular read before a particular write.

57. (Previously presented) The computer readable storage medium of claim 52, wherein the lock is rebiasable to another thread during course of a computation that employs the lock.

58. (Currently amended) A computer implemented method, comprising:

a computer processor ~~that supports out-of-order instruction execution~~ performing:

biasing a lock to a first thread of execution, wherein the biasable lock is concurrently accessible to a plurality of threads and is biased to at most one of the plurality of threads at a given time, wherein the biasable lock includes a quick-lock field; and

subsequently acquiring the lock for, or by, the first thread with computational overhead less than for a second thread to which the lock is not currently biased, wherein said acquiring is performed free of atomic read-modify-write operations and comprises:

storing a value in the quick-lock field, the value indicating that the lock is held by the first thread; and

subsequent to said storing, determining whether the lock is still biased to the first thread, wherein said acquiring completes successfully with computational overhead less than for the second thread, if the lock is determined to still be biased to the first thread

~~and of explicit memory barrier operations executable to constrain the processor from performing at least one type of instruction reordering~~.

59. (Canceled)

60. (Canceled)

61. (Previously presented) The computer implemented method of claim 58, further comprising: rebiasing the lock to a third thread of execution.

62. (Currently amended) A computer system, comprising:

a processor coupled to one or more other processors ~~and configured to perform out of order instruction execution~~;

a memory coupled to the processor, the memory storing program instructions executable by the processor to implement:

a biasable lock that is concurrently accessible to a plurality of threads on the multithreaded system and is biased to at most one of the plurality of threads at a given time;

acquisition and release sequences that, when executed by a bias-holding thread on the processor, wherein the biasable lock is biased to the bias-holding thread, do not comprise executing atomic read-modify-write operations and wherein the acquisition sequence comprises:

storing a value in the quick-lock field, the value indicating that the biasable lock is held by the bias-holding thread; and

subsequent to said storing, determining whether the biasable lock is still biased to the bias-holding thread, wherein the acquisition sequence completes successfully and free of atomic read-modify-write operations if the biasable lock is determined to still be biased to the bias-holding thread

~~or explicit memory barrier operations executable to constrain the processor from performing at least one type of instruction reordering~~.

63. – 65. (Canceled)

66. (Previously presented) The computer system of claim 62, further comprising: a rebiasing sequence, executable by the processor to bias the biasable lock to another thread on the system instead of to the bias-holding thread.

67. (Previously presented) The computer system of claim 62, wherein the memory is implemented as at least one computer readable storage medium selected from the set of a disk, tape or other magnetic, optical, or electronic storage medium.

68. – 71. (Cancelled)

72. (Currently amended) A computer implemented method of making a single computer program product suitable for efficient execution as both a single-threaded computation and a multi-threaded computation on one or more processors ~~capable of out-of-order execution~~, the method comprising:

structuring a computation as a multithread computation;

mediating at least some sources of contention in the multithreaded computation using a bias[[ ]]able locking mechanism, wherein the biasable locking mechanism is concurrently accessible to a plurality of threads on the one or more processors and is biased to at most one of the plurality of threads at a given time, wherein the biasable locking mechanism includes a quick-lock field;

wherein ~~the~~ said mediating includes providing at least two acquisition sequences including a fast path acquisition sequence for a bias-holding thread to which the biasable lock has been biased and a second acquisition sequence, the fast path acquisition sequence being optimized relative to the second acquisition sequence, wherein the fast path acquisition sequence comprises:

storing a value in the quick-lock field, the value indicating that the biasable locking mechanism is held by the bias-holding thread; and

subsequent to said storing, determining whether the biasable locking mechanism is still biased to the bias-holding thread, wherein the fast path acquisition sequence completes successfully and free of atomic read-modify-write operations if the biasable locking mechanism is determined to still be biased to the bias-holding thread;

introducing the instances of the biasable locking mechanism into program code;

compiling the program code; and

encoding the compiled program code, including the instances of the biasable locking mechanism, in a computer program product.

73. (Previously presented) The method of claim 72, wherein the encoding includes transferring the compiled program code onto at least one computer readable storage medium selected from the set of a disk, tape or other magnetic, optical, or electronic storage medium.